EPFL - Information, Calcul et Communication - Partie théorique

Série 5: Solutions

1 Qu'est-ce qui est difficile?

Problème a) Déterminer si un nombre donné x est premier est a priori difficile: une façon directe de faire est de chercher à savoir si le contraire est vrai, i.e. si x est divisible par un nombre premier plus petit que lui. Par exemple, pour savoir si 29 est premier, on essaye de le diviser par 2, puis 3, puis 5, puis 7 (et c'est tout: en effet, on n'a pas besoin de tester tous les nombre premiers jusqu'à 29, mais seulement ceux jusqu'au nombre premier venant juste après $\sqrt{29}$; vous devinez pourquoi?). Bref, si le nombre x est composé lui-même de n chiffres, on va devoir a priori tester tous les nombres premiers plus petits que $\sqrt{x}$, qui est lui composé de $n/2$ chiffres environ, et ça fait du monde (de l'ordre de $10^{n/2}$ possibilités, donc un nombre exponentiel en n)! Notez cependant qu'il existe aujourd'hui des algorithmes sensiblement plus performants qui résolvent le problème en un temps polynomial en n.

Problème b) Additionner deux nombres entiers x et y, chacun composé de n chiffres, est par contre facile et ne vous prendra que n opérations en moyenne.

Problème c) Multiplier deux nombres entiers x et y, chacun composé de n chiffres, est aussi facile, mais demande a priori de l'ordre de $n \times n = n^2$ opérations, si on suit l'algorithme simple que vous avez appris à l'école primaire. Là encore, aujourd'hui, il existe des algorithmes qui permettent de réduire le nombre d'opérations à effectuer (sans toutefois descendre plus bas que le nombre d'opérations à effectuer pour une simple addition (problème b)).

Problème d) Il existe de nombreux algorithmes pour trier une liste de n nombres arbitraires dans l'ordre croissant. Parmi les meilleurs, le tri par fusion nécessite de l'ordre de $n \cdot \log_2(n)$ opérations.

Problème e) Le meilleure manière pour trouver un nombre choisi au hasard dans une liste de n nombres est d'utiliser l'algorithme de recherche par dichotomie qui nécessite de poser de l'ordre de $\log_2(n)$ questions seulement.

Problème f) Déterminer le meilleur coup à jouer aux échecs pour obtenir la meilleure position possible n coups plus tard reste un problème très difficile. En supposant qu'on puisse jouer une dizaine de coups " raisonnables " à chaque fois, on reste avec 10^n possibilités à analyser! C'est l'incroyable puissance de calcul des ordinateurs modernes, et bien sûr aussi toute une série d'innovations dans le monde des algorithmes, qui ont permis de réaliser des ordinateurs capables d'évaluer un grand nombre de possibilités, de manière à finalement battre les meilleurs joueurs d'échecs humains.

Donc le classement final est (du plus facile au plus difficile): **e), b), d), c), a), f)**.

2 Coloration de graphes

a) Pour le graphe de gauche, la réponse est oui; pour celui de droite, la réponse est non.

b) L'algorithme est le suivant: on part d'un sommet (n'importe lequel), qu'on colorie d'une couleur (disons vert). Après cela, on colorie tous ses voisins de l'autre couleur (disons rouge), puis on essaye de colorer tous les voisins des voisins à nouveau en vert, etc. Si, en procédant ainsi, on se retrouve à un moment donné avec deux sommets voisins de même couleur, alors on sait qu'une coloration de tout le graphe avec seulement deux couleurs n'est pas possible. Si par contre, on arrive jusqu'au bout sans problème, alors on a trouvé une solution au problème.

c) Le nombre d'opérations à effectuer pour exécuter l'algorithme ci-dessus est le suivant: au pire, on doit parcourir tous les n sommets du graphe, et à chaque sommet, on doit vérifier qu'aucun des voisins n'a déjà été colorié dans la même couleur que celle qu'on est en train d'utiliser pour colorer le sommet. Comme chaque sommet peut avoir jusqu'à $n-1$ voisins (ce sera le cas pour certains sommets dans certains graphes), on risque

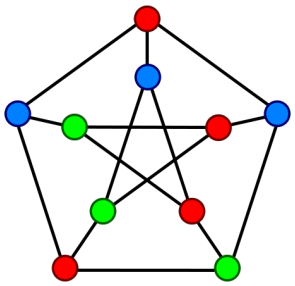
au pire d'effectuer $\Theta(n^2)$ opérations en tout (mais bien sûr, il se peut qu'on puisse conclure beaucoup plus vite que ça dans certains cas).

Quoi qu'il en soit, le fait qu'on n'ait besoin au pire que de $\Theta(n^2)$ opérations pour résoudre le problème implique que le problème de la coloration d'un graphe avec deux couleurs est dans la classe P (c'est un problème soluble en un temps polynomial en le nombre de variables).

d) Pour vérifier si une coloration donnée avec k couleurs fonctionne, on parcourt simplement tous les sommets du graphe, et pour chacun des sommets, on vérifie qu'aucun de ses voisins n'est colorié de la même couleur que le sommet lui-même. Comme il y a n sommets et que chacun des sommets peut avoir au pire $n-1$ voisins, le nombre total d'opérations nécessaires pour cette vérification est à nouveau un $\Theta(n^2)$. Le problème de la coloration d'un graphe avec k couleurs est donc dans la classe NP (c'est un problème tel que si on nous donne une solution du problème, alors il est possible de vérifier en un temps polynomial en le nombre de variables si cette solution est correcte ou non).

e*) Sans indication de départ, trouver une coloration avec trois couleurs qui fonctionne pour un graphe donné est a priori (beaucoup) plus compliqué. Bien sûr, s'il se trouve qu'une coloration avec deux couleurs fonctionne, comme dans le cas du graphe de gauche, alors automatiquement on sait dans ce cas qu'une coloration avec trois couleurs fonctionne également (il suffit de remplacer la couleur d'un des sommets avec la troisième couleur, par exemple); en fait, beaucoup de colorations avec trois couleurs fonctionnent dans ce cas.

Par contre, si aucune coloration du graphe avec deux couleurs ne fonctionne (comme dans le cas du graphe de droite), alors tout se complique pour le cas de trois couleurs. Il se trouve que pour le graphe de droite, une telle coloration existe:



Commet le trouver? (comment l'avez-vous trouvé vous-même?) On peut bien sûr essayer en tâtonnant et espérer avoir de la chance... On peut aussi essayer toutes les possibiliés, comme suggéré dans l'énoncé de l'exercice, mais le nombre d'opérations nécessaires sera alors de 3^n, donc exponentiel en le nombre de variables... Est-il possible de faire mieux? A l'heure actuelle, la réponse à cette question est: on ne sait pas s'il existe un algorithme capable de résoudre ce problème en un temps polynomial en n (ceci reviendrait à prouver que P=NP).

3 Pour le plaisir: Sudoku et Mastermind*

a) Il est toujours facile de vérifier qu'un Sudoku entièrement rempli satisfait les conditions mentionnées dans l'énoncé. Plus précisément, il faut vérifier que chaque ligne, colonne et sous-tableau carré (contenant chacune et chacun n chiffres) contient des éléments tous différents, ce qui prend de l'ordre de n^2 opérations à chaque fois (cf. début du cours). Au total, il y a n lignes, n colonnes et n sous-tableaux, donc le nombre total d'opérations à effectuer est de l'ordre de n^3, ce qui est bien polynomial en n. Le problème appartient donc à la classe NP. Par contre, à partir d'un Sudoku partiellement rempli, trouver la solution peut s'avérer dans certains cas être très compliqué. On ne sait pas à l'heure actuelle s'il existe un algorithme de résolution dont la complexité temporelle soit polynomiale en n. Donc on ne sait pas si le problème est dans P.

b) Même chose ici: si on nous montre la solution, il est facile de vérifier que chacune des n couleurs ou lettres satisfait aux indications données par l'adversaire. Le problème fait donc aussi partie de la classe NP. Mais trouver la solution à partir des indications données peut s'avérer sensiblement plus compliqué (imaginez que $n = 1'000...$). Donc on ne sait pas si le problème est dans P (*Note:* Dans l'exemple, la solution est "SUD").